CSc 174 Database Management Systems

10. Relational Database Design Algorithms

Ying Jin
Computer Science Department
California state University, Sacramento

1. Properties of Relational Decompositions

Properties

- Attribute preservation
- Dependency preservation property
- Lossless join property

Universal Relation

- **Universal Relation Schema**:

  a relation schema $R=\{A_1, A_2, \ldots, A_n\}$ that includes all the attributes of the database.

- **Universal relation assumption**

  every attribute name is unique

- **Decomposition**:

  decompose R into $D = \{R_1, R_2, \ldots, R_m\}$, by using the functional dependencies.

Attribute Preservation

- **Attribute preservation**

  Each attribute in R will appear in at least one relation schema R_i in the decomposition, so that no attributes are "lost".

Dependency Preservation Property -Informally

- Informally:

  Each function dependency $X \rightarrow Y$ specified in F, either

  (1) appeared directly in one of the relation schemas Ri in the decomposition D, or

  (2) could be inferred from the dependencies that appear in some Ri.

- What is F^+ (Closure of F)?

Project of F on R - Definition

- Definition:

  Given a set of dependencies F on R,
  the **projection** of F on R_i, denoted by $\pi_{R_i}(F)$, where $R_i \subseteq R$, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i.

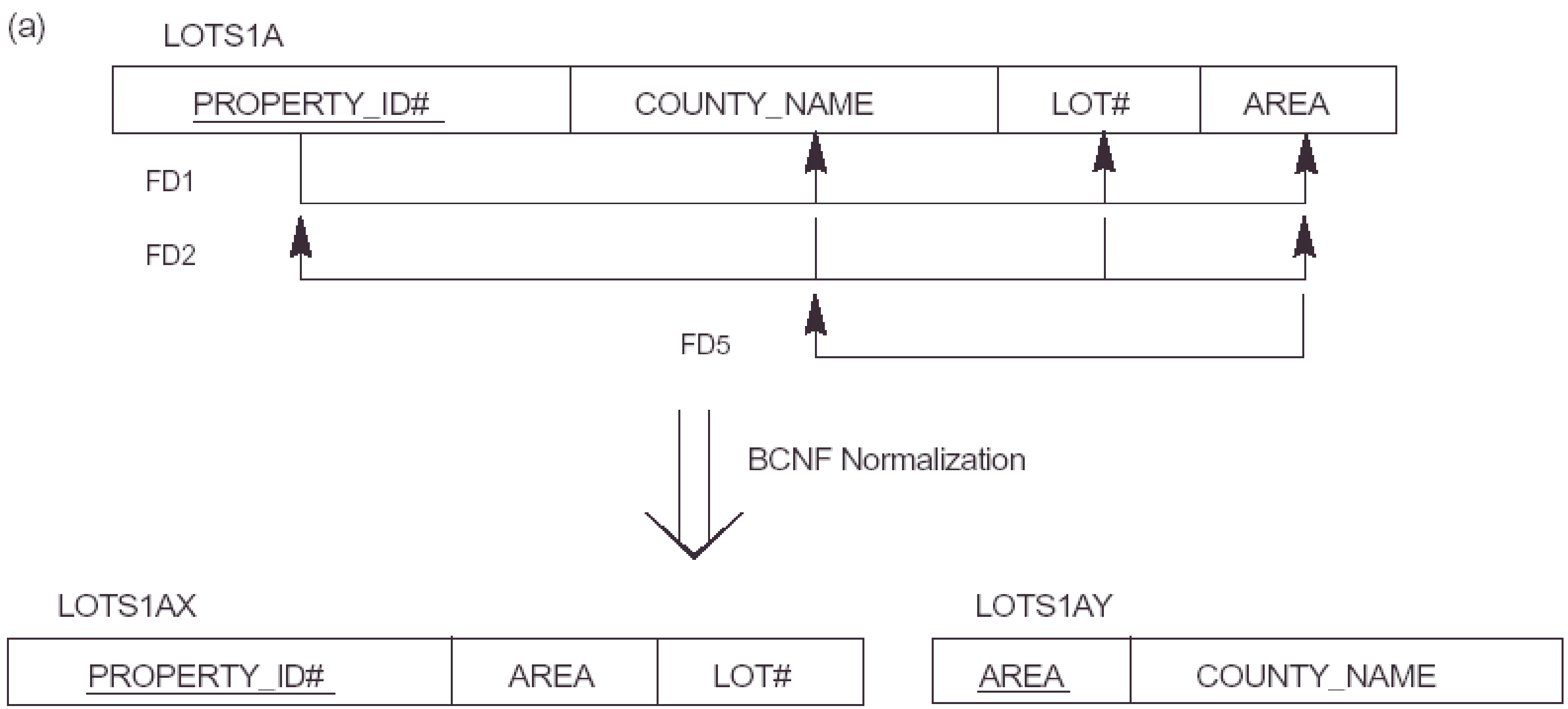
all their left- and right-hand-side attributes are in R_i.

Dependency Preservation Property

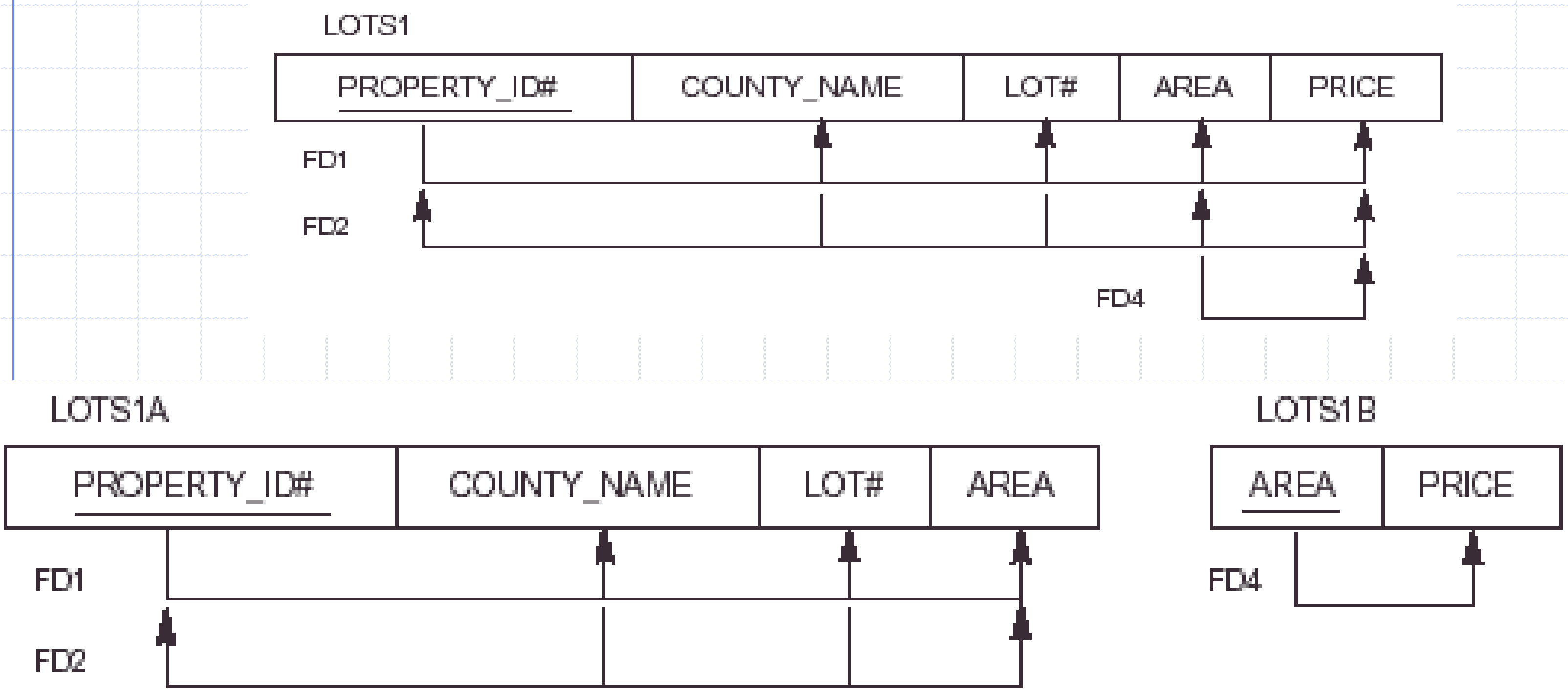
- Dependency Preservation Property definition: a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R is **dependency-preserving** with respect to F if the union of the projections of F on each R_i in D is equivalent to F; that is, ($(\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$
- **Claim 1:** It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation R_i in D is in 3nf.

Not Dependency-Preservation Example

- Which fd is lost?



Dependency-Preservation Example



Lossless (Non-additive) Join Property

- No addition of spurious tuples.
- Definition

  Lossless join property: a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R has the **lossless (nonadditive) join property** with respect to the set of dependencies F on R if, for *every* relation state r of R that satisfies F, the following holds, where * is the natural join of all the relations in D:

$$* (\pi_{R1}(r), ..., \pi_{Rm}(r)) = r$$

- **Note**: *lossless* refers to *loss of information,* not to loss of tuples.

Algorithm for Lossless (Non-additive) Join Property of a Decomposition

Testing for Lossless Join Property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set $S(i,j) := b_{ij}$ for all matrix entries. (each b_{ij} is a distinct symbol associated with indices (i,j)).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
      {if (relation R_i includes attribute A_j) then set $S(i,j) :=$ a_j;};};

(each a_j is a distinct symbol associated with index (j))

Algorithm for Lossless (Non-additive) Join Property of a Decomposition (Cont.)

4. Repeat the following loop until a *complete loop execution* results in no changes to *S*

   {for each functional dependency $X \rightarrow Y$ in *F*

   {for all rows in *S which have the same symbols* in the columns corresponding to attributes in *X*

   {make the symbols in each column that correspond to an attribute in *Y* be the same in all these rows as follows: if any of the rows has an "*a*" symbol for the column, set the other rows to that *same* "*a*" symbol in the column; If no "a" symbol exists for the attribute in any of the row, choose one of the "b" symbols that apperas in one of the rows for the attribute and set the other rows to that same "b" symbol in the column };};};

5. If a row is made up entirely of "*a*" symbols, then the decomposition has the lossless join property; otherwise it does not.

Lossless (nonadditive) join test for *n*-ary decompositions – Example

- Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

EMP

SSN	ENAME

PROJECT

PNUMBER	PNAME	PLOCATION

WORKS_ON

SSN	PNUMBER	HOURS

(c) R={SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS} $D=\{R_1, R_2, R_3\}$

R_1=EMP={SSN, ENAME}

R_2=PROJ={PNUMBER, PNAME, PLOCATION}

R_3=WORKS_ON={SSN, PNUMBER, HOURS}

F={SSN→{ENAME;PNUMBER→{PNAME, PLOCATION} ;{SSN,PNUMBER}→HOURS}

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(original matrix S at start of algorithm)

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	$\cancel{b_{32}}$ a_2	a_3	$\cancel{b_{34}}$ a_4	$\cancel{b_{35}}$ a_5	a_6

(matrix S after applying the first two functional dependencies - last row is all "a" symbols, so we stop)

Binary Decomposition

- **Binary Decomposition**: decomposition of a relation R into two relations.
- **PROPERTY LJ1 (lossless join test for binary decompositions):** A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
  - The f.d. $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or
  - The f.d. $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+.

Successive Lossless Join Decomposition

- **Claim 2 (Preservation of non-additivity in successive decompositions):**

  If a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R has the lossless (non-additive) join property with respect to a set of functional dependencies F on R, and if a decomposition $D_i = \{Q_1, Q_2, ..., Q_k\}$ of R_i has the lossless (non-additive) join property with respect to the projection of F on R_i, then the decomposition $D_2 = \{R_1, R_2, ..., R_{i-1}, Q_1, Q_2, ..., Q_k, R_{i+1}, ..., R_m\}$ of R has the non-additive join property with respect to F.

2. Algorithms for Relational Database Schema Design

Relational Synthesis into 3NF with Dependency Preservation (*Relational Synthesis Algorithm)*

- Does not guarantee the lossless join property

Input: A universal relation R and a set of functional dependencies F on the attributes of R.

1. Find a minimal cover G for F (use Algorithm 10.2);
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_k$ are the only dependencies in G with X as left-hand-side (X is the *key* of this relation) ;
3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

Relational Decomposition into BCNF with Lossless (non-additive) join property

- No guarantee of dependency preservation

Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. Set D := {R};
2. While (there is a relation schema *Q* in *D* that is not in BCNF) do {

   choose a relation schema *Q* in *D* that is not in BCNF;

   find a functional dependency $X \rightarrow Y$ in *Q* that violates BCNF;

   replace *Q* in *D* by two relation schemas $(Q - Y)$ and $(X \cup Y)$;

   };

Relational Synthesis into 3NF with Dependency Preservation *and* Lossless (Non-Additive) Join Property

- **Input:** A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. Find a minimal cover *G* for *F* (*Use Algorithm 10.2*).
2. For each left-hand-side *X* of a functional dependency that appears in *G*, create a relation schema in *D* with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_k$ are the only dependencies in *G* with *X* as left-hand-side (*X* is the *key* of this relation).
3. If none of the relation schemas in *D* contains a key of *R*, then create one more relation schema in *D* that contains attributes that form a key of *R*. (*Use Algorithm 11.4a to find the key of R*).
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a project of another relation S in the schema.

Algorithm: Finding a Key *K* for *R* Given a set *F* of Functional Dependencies

Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. Set $K := R$.
2. For each attribute *A* in *K* {

    compute $(K - A)^+$ with respect to *F*;

    If $(K - A)^+$ contains all the attributes in *R*,

    then set $K := K - \{A\}$; }

Discussion of Normalization Algorithms

Problems:

- The database designer must first specify *all* the relevant functional dependencies among the database attributes.
- It is not always possible to find a decomposition into relation schemas that preserves dependencies and allows each relation schema in the decomposition to be in BCNF (instead of 3NF as in Algorithm 11.4).
- Many different designs arise corresponding to the same set of functional dependencies, depending on the order in which such dependencies are considered.
- Some design may be superior, or undesirable.

Algorithms for Relational Database Schema Design

	Input	Output	Properties/Purpose	Remarks
	A decomposition D of R and a set F of functional dependencies	Boolean result: yes or no for lossless join property	Testing for lostless join decomposition	
	Set of functional dependencies F	A set of relations in 3NF	Dependency preservation	No guarantee of satisfying lossless join property
	Set of functional dependencies F	A set of relations in BCNF	Lossless join decomposition	No guarantee of dependency preservation
	Set of functional dependencies F	A set of relations in 3NF	Lossless join **and** dependency preserving decomposition	May not achieve BCNF
	Relation schema R with a set of functional dependencies F	Key K of R	To find a key K (which is a subset of R)	The entire relation R is always a default superkey

These slides are based on the textbook:

R. Elmasri and S. Navathe, *Fundamentals of Database System*, 7th Edition, Addison-Wesley.